

Cameras

Geometric Calibration

Summary

- Geometric model of a camera
 - intrinsic & extrinsic parameters
 - distortion parameters
- Estimation of the model parameters
- Calibration procedures

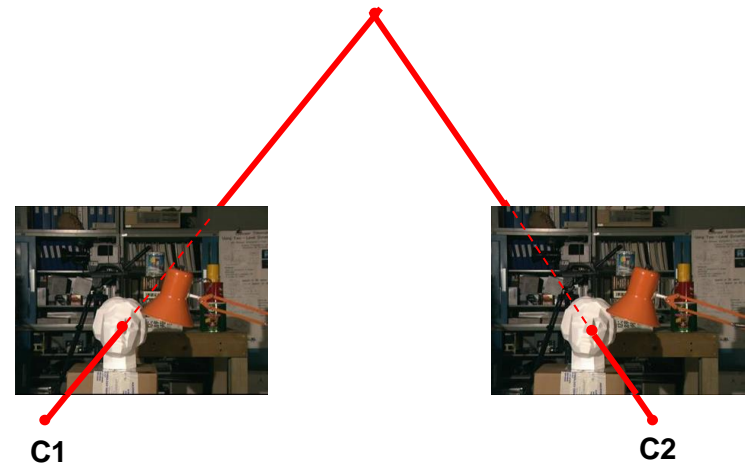
Camera calibration

- Camera geometric model
 - How can we represent the image formation process ?
 - lens, image sensor, "pixelization", ...
 - What is the relation between the camera's natural units (pixels) and the units of the physical world (e.g., meters) ?
 - *Only interested in the geometry of image formation (no photometric concerns)*
- Determination of the parameters of the model
 - Requires imaging an object with some points whose 3D position is accurately known
 - How many points are needed ?
 - we'll see in the following

Why calibrate?

An example: stereo vision

- Basic principle
 - triangulation
- Main steps
 - acquire 2 images from different viewpoints
 - match points between the images
 - for each point of a matching pair determine its line-of-sight intersect the 2 lines-of-sight
- To obtain the equations of the 2 lines-of-sight we need to build a geometric model of the cameras
⇒ geometric calibration of the camera



Note: in fact, the images are formed behind points C1 and C2 and are inverted ...

Camera model

- Image formation model
 - diaphragm model (pinhole)
 - lens model (thin lens)
 - perspective projection equation
- (Geometric transformations in homogeneous coordinates)
- Camera model
 - relationship between
world - (x,y,z) - and pixel - (i,j) - coordinates
 - perspective projection matrix (/ DLT - direct linear transform)
 - intrinsic, extrinsic and distortion parameters

Pinhole camera

- A simple camera model is the pinhole camera model
- Light passes through a small hole and falls on an image plane
- The image is inverted and scaled
- Early cameras used this approach

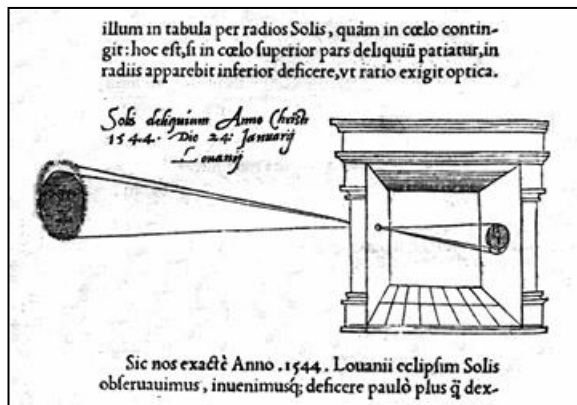
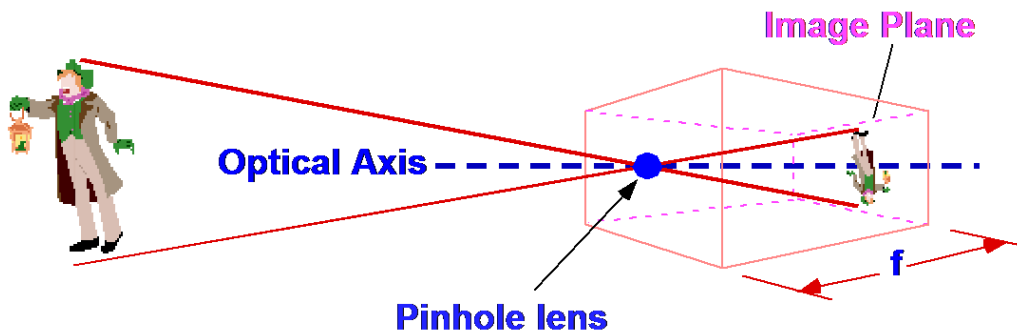
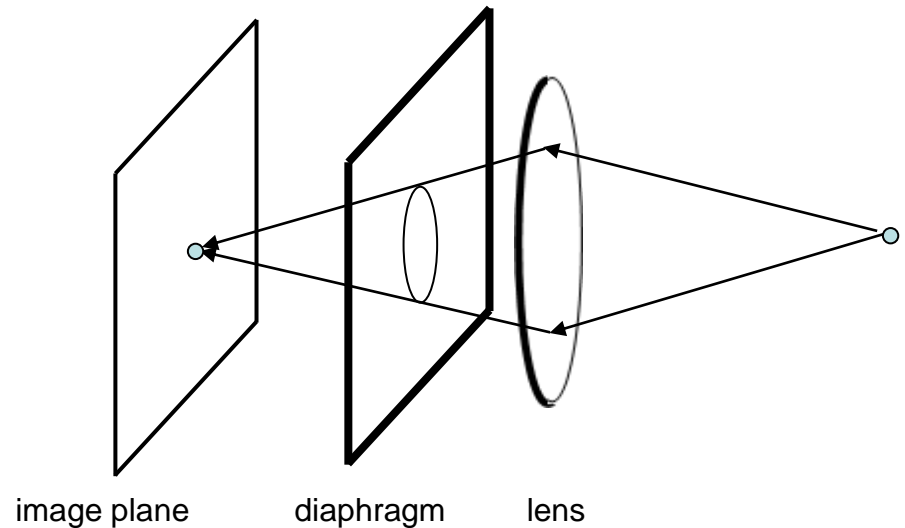
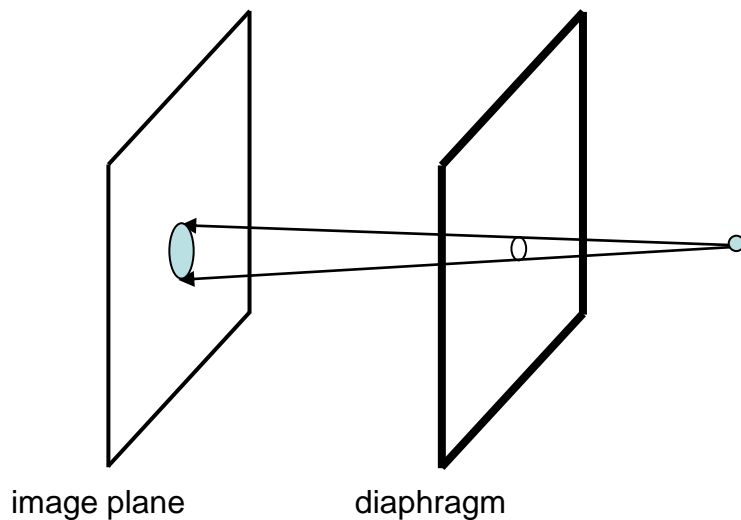


Image acquired with pinhole camera
[Robert Kosara](#)

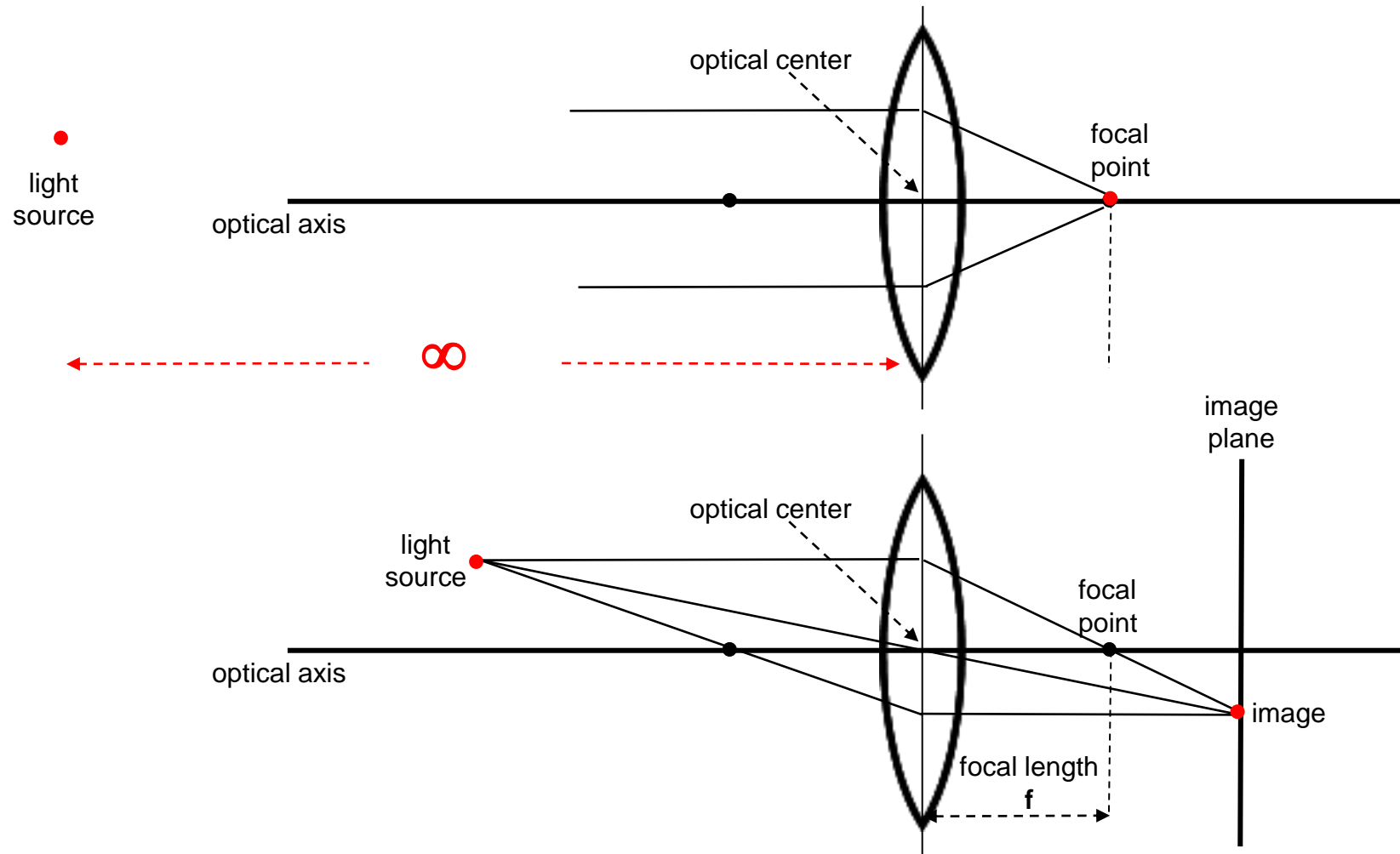
Real camera

- Real cameras do not have pinholes
- A pinhole lets very little light in, leading to long exposure time
- Larger holes lead to blurred images
- A lens can make a large hole act like a small one

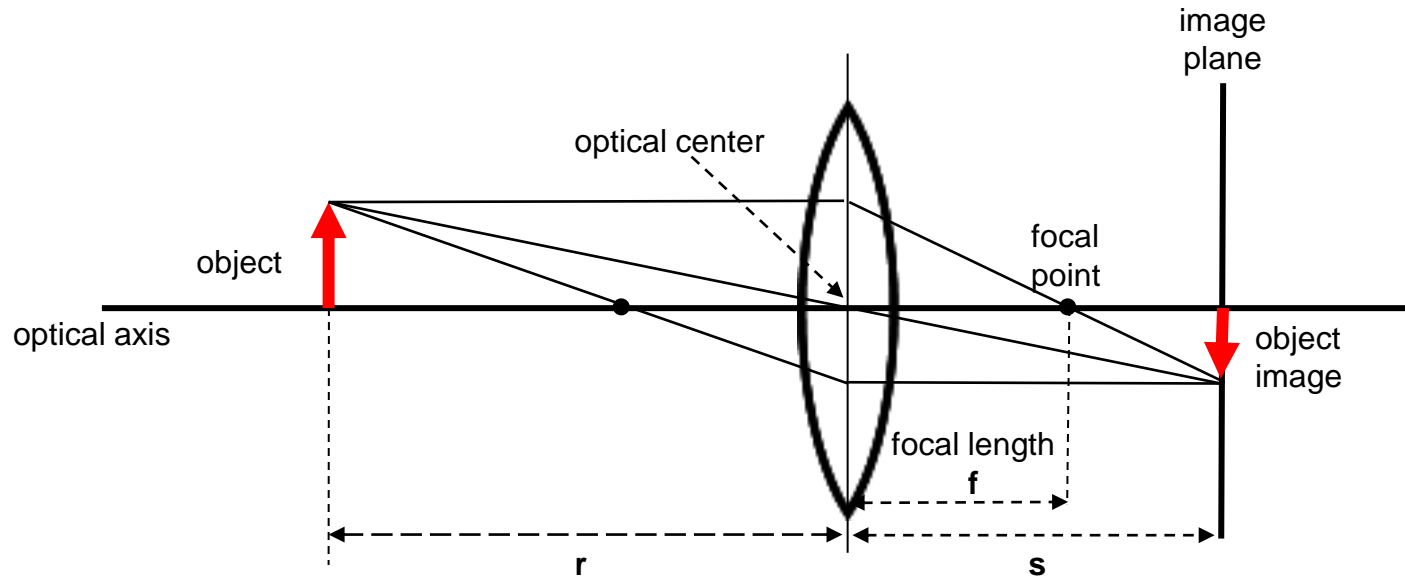


Imaging through a lens

- A light source at an infinite distance is focused on the focal point
- If the light source moves closer, the light rays collect behind the focal point



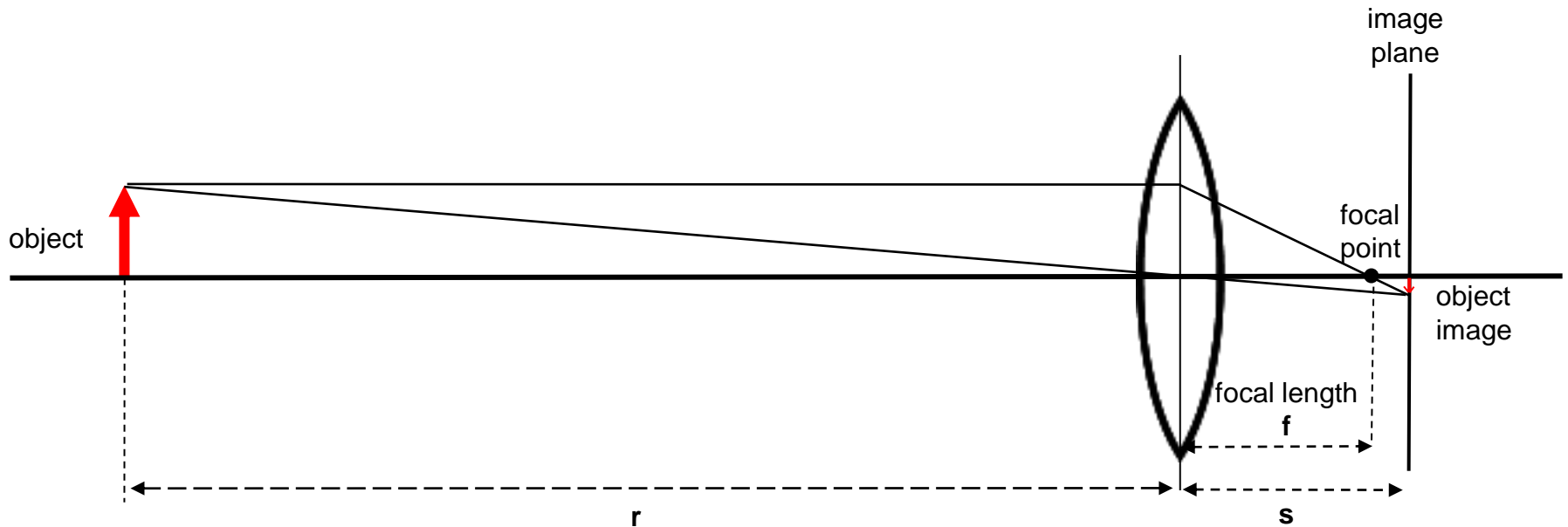
Thin lens equation



$$\frac{1}{r} + \frac{1}{s} = \frac{1}{f}$$

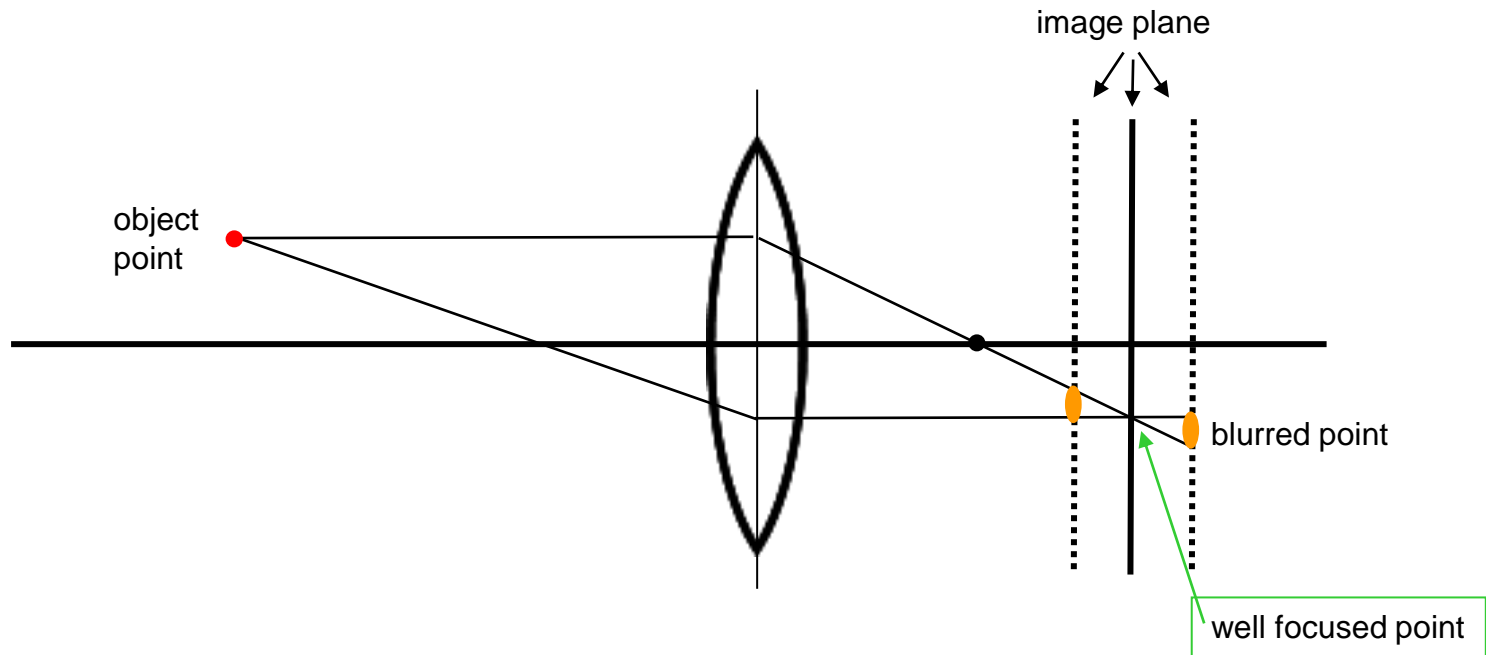
**Gauss
lens
equation**

Image formation model



- When r is large enough ($r \gg f$) $\Rightarrow s \approx f$

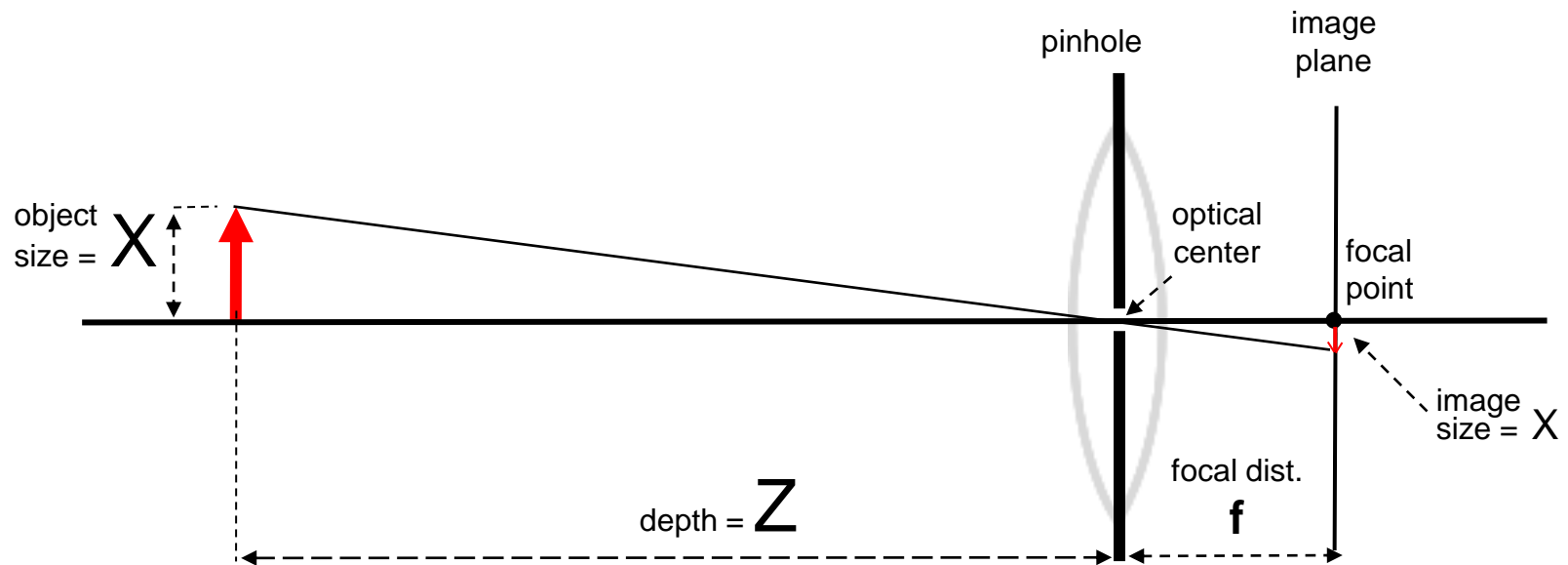
Image formation



- When the image plane is not on the right place the image is defocused
 - points are projected as blurred circles

Image formation model

- Considering
 - objects distant enough
 - pinhole model
- The image is formed at the focal distance and is always focused (...!)

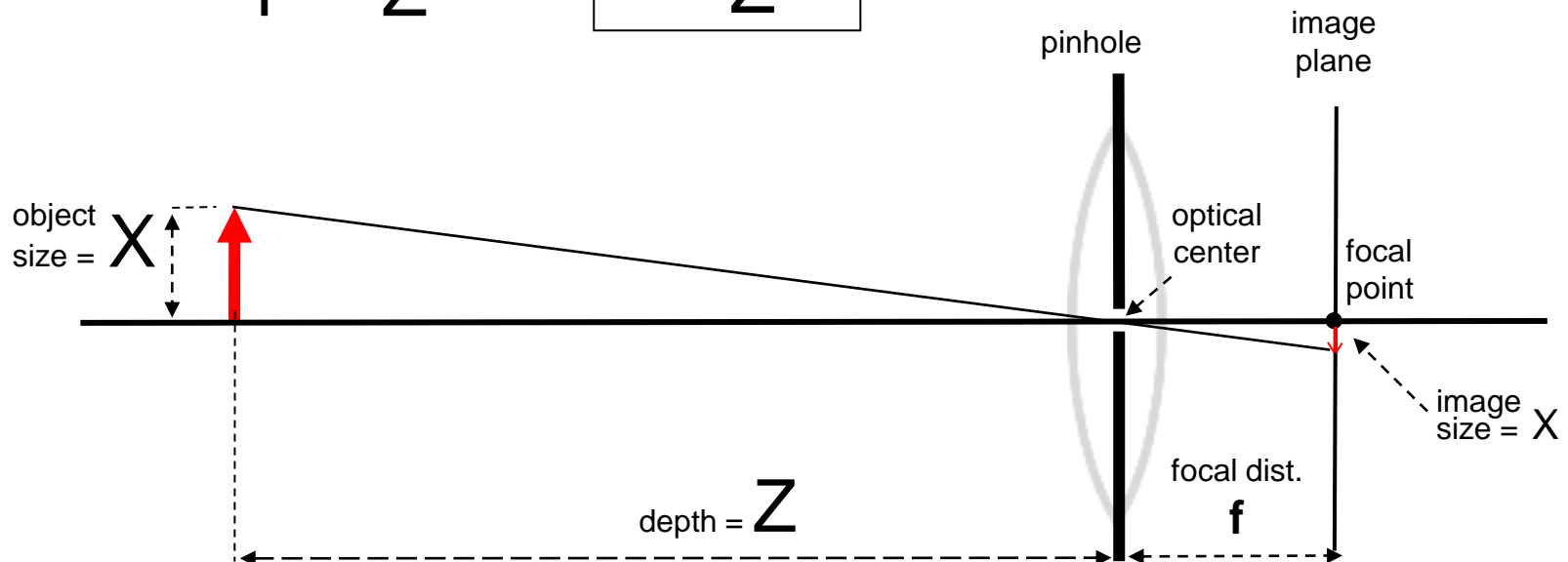


Perspective projection

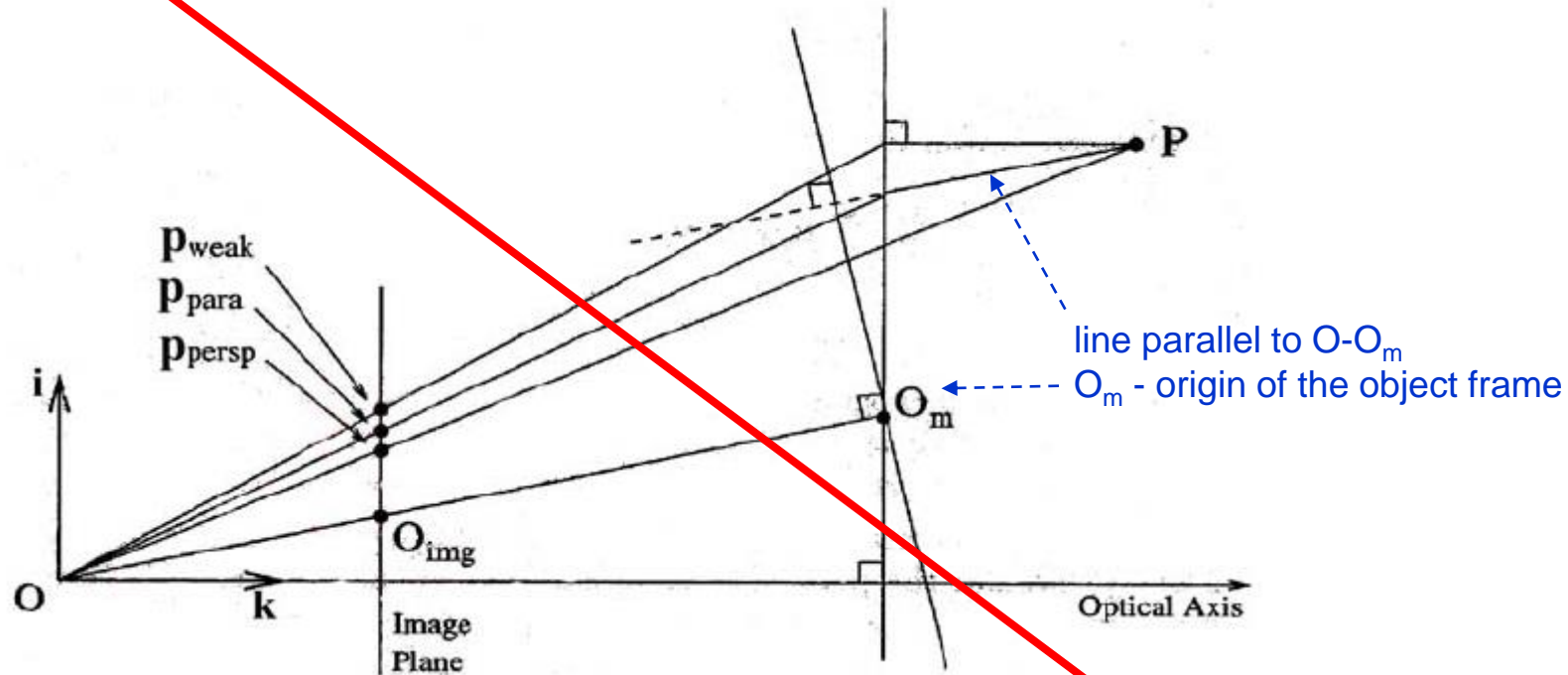
- The image is a scaled version of the object
- The scaling depends on the distance to the object, Z (perspective projection)



$$\frac{x}{f} = \frac{X}{Z} \Rightarrow \boxed{x = \frac{f}{Z} X}$$



Perspective, paraperspective and weak perspective camera models



Geometry of the perspective, paraperspective and weak perspective camera models

Weak perspective projection

- If the “object” extends through a small depth, ΔZ , when compared to the mean distance Z_0 , the scaling can be considered constant ($= f / Z_0$) (weak perspective projection)

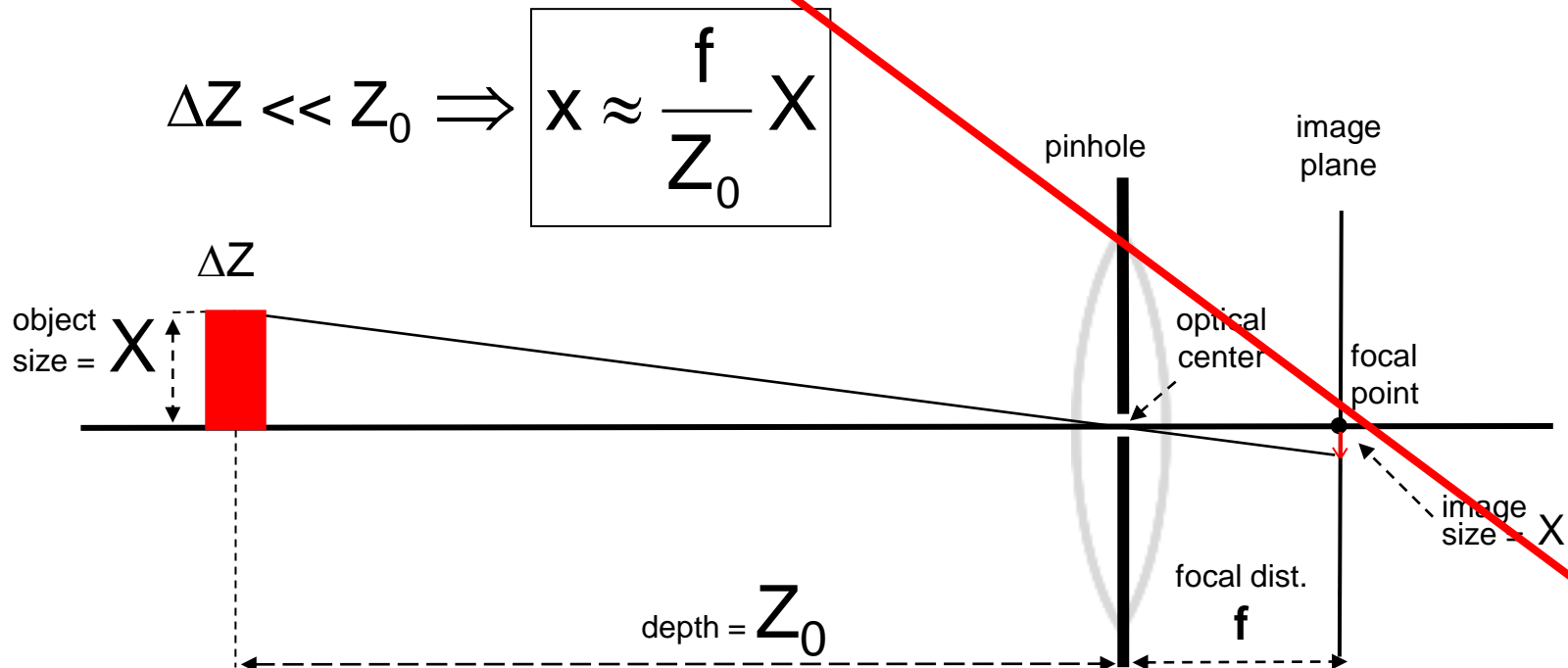


Image formation model

- The image plane is traditionally drawn in front of the optical center
- Avoids signal inversion, between object and image coordinates

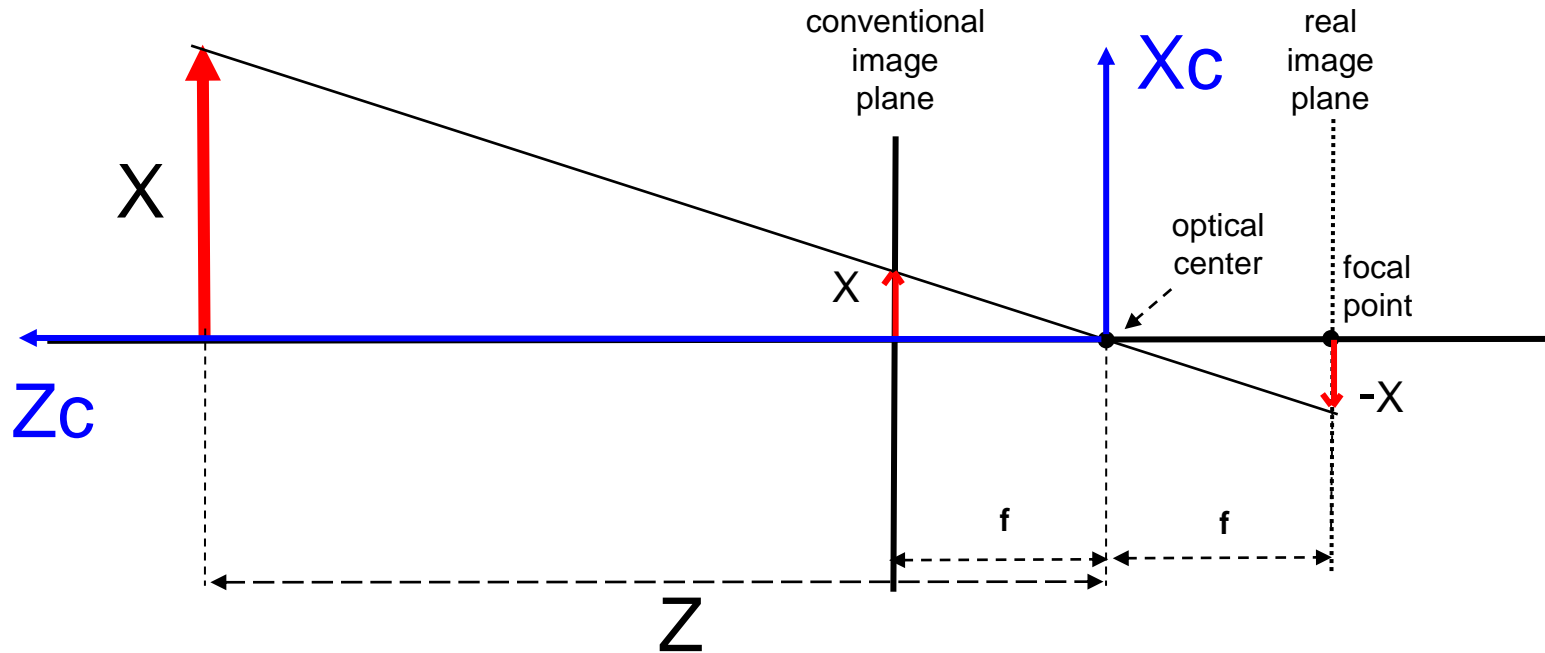
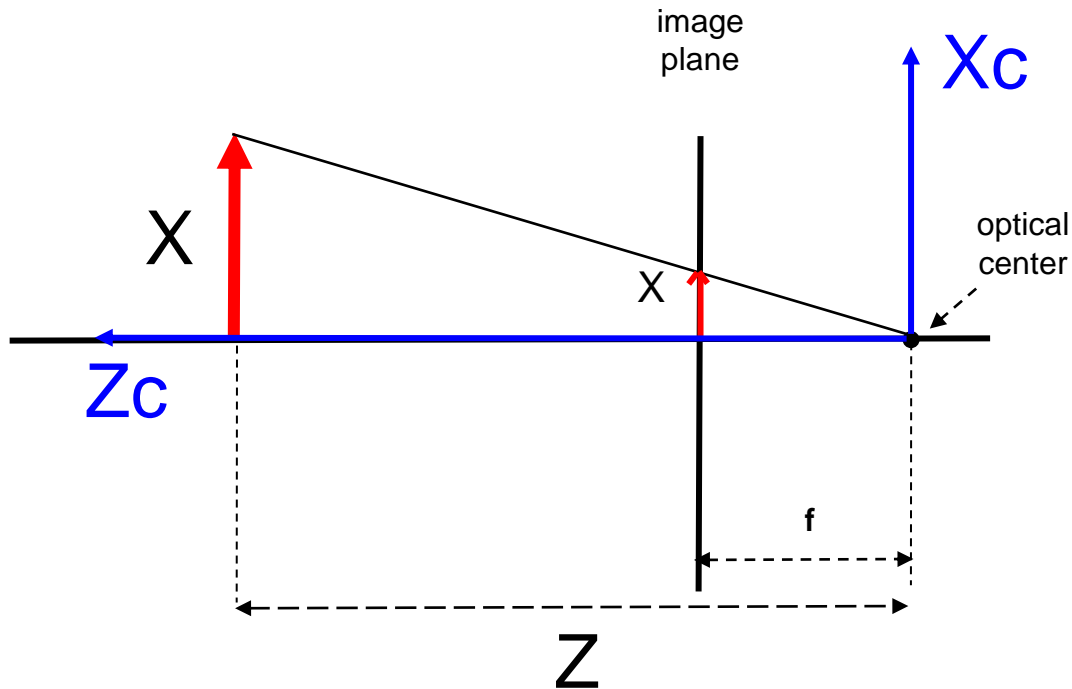
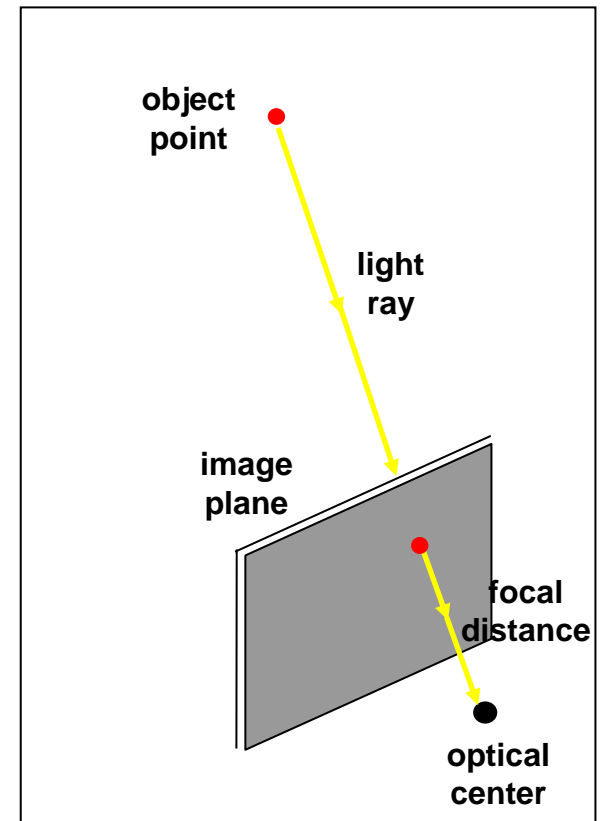


Image formation model



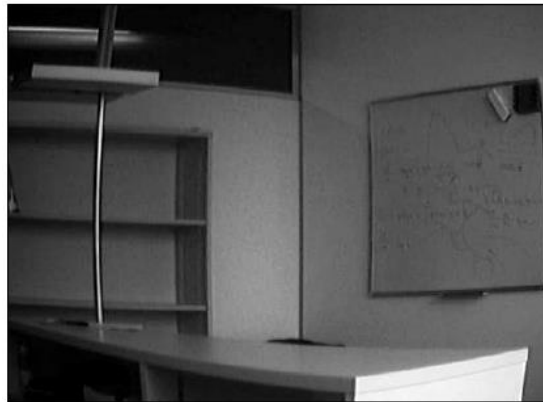
$$x = \frac{f}{Z} X$$

$$y = \frac{f}{Z} Y$$

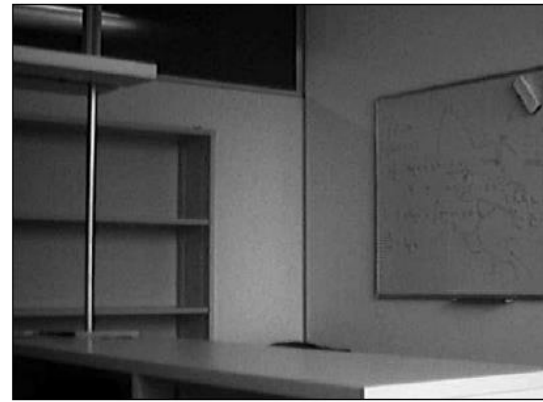


Lens distortion

- The previous image formation model does not take into account distortions due to lens construction (ex. fish eye lenses) or bad quality lenses
- Distortion parameters can be estimated and image distortion can be corrected
- *Distortion will be addressed later.*



Distorted image
(barrel distortion)



Corrected image

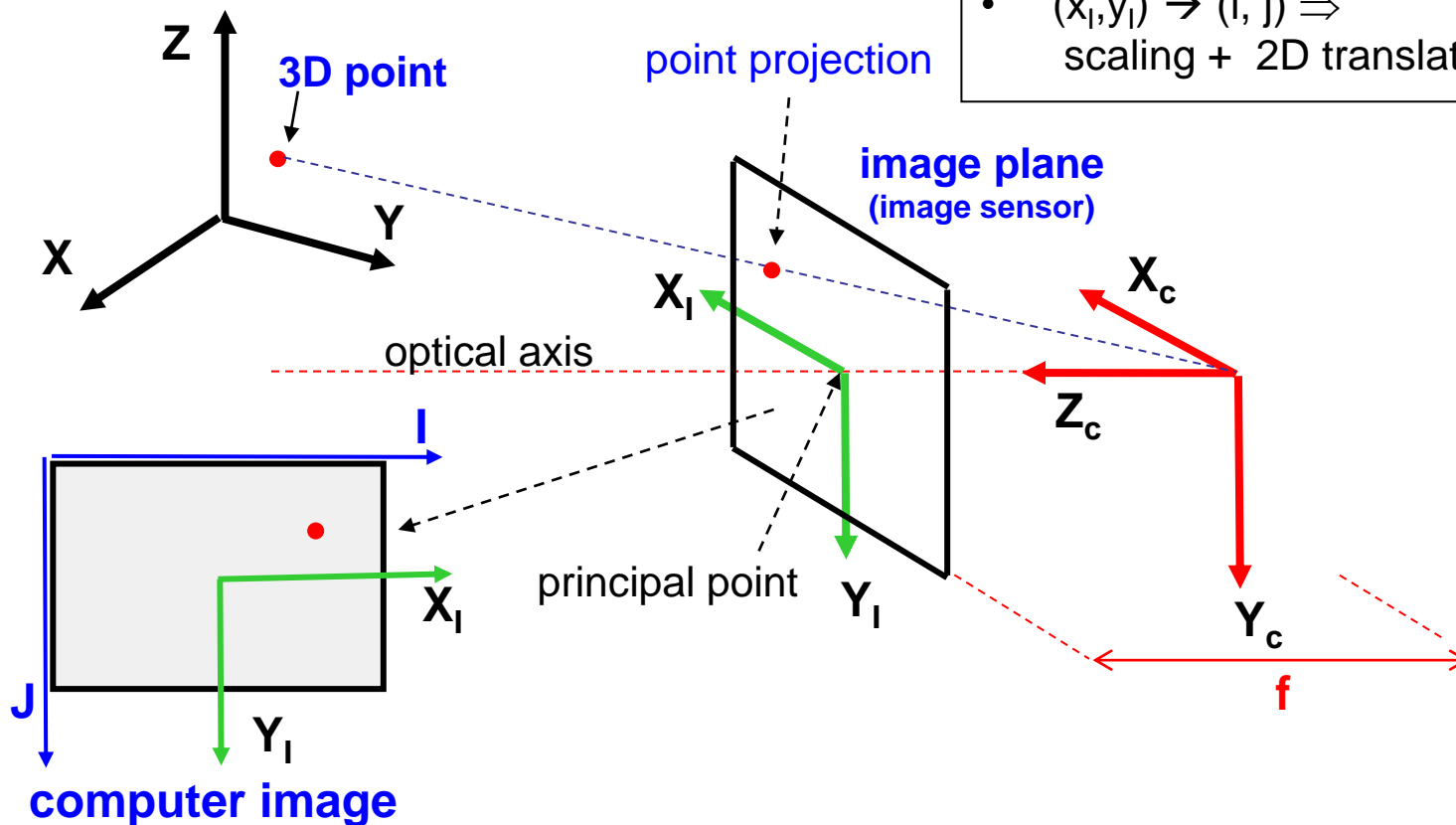
Camera model

Reference frames

- World - (X, Y, Z)
- Camera - (X_C, Y_C, Z_C)
- Sensor - (X_I, Y_I)
- Image - (I, J)

Coordinate transformations

- $(x, y, z) \rightarrow (x_C, y_C, z_C) \Rightarrow$
3D rotation + 3D translation
- $(x_C, y_C, z_C) \rightarrow (x_I, y_I) \Rightarrow$
perspective projection
- $(x_I, y_I) \rightarrow (i, j) \Rightarrow$
scaling + 2D translation



Homogeneous coordinates



Homogeneous coordinates

- Homogeneous coordinates are a key tool for
 - 3D computer vision
 - 3D computer graphics
 - Modeling robotic manipulators
 - ...
- They allow us to transform between reference frames with a single matrix multiplication
 - in “normal” Euclidean coordinates, rotation is expressed by a matrix multiplication but translation is expressed by an addition ...

Homogeneous coordinates

- (N+1)-dimensional notation for points in N-dimensional Euclidean space

Inhomogeneous (“normal”) coordinates

$$v = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Example:

$$v = \begin{bmatrix} 4 \\ -2 \\ 5 \end{bmatrix} \quad \begin{array}{l} \nearrow w=1 \\ \searrow w=-2 \end{array} \quad \begin{array}{l} v = \begin{bmatrix} 4 \\ -2 \\ 5 \\ 1 \end{bmatrix} \\ v = \begin{bmatrix} -8 \\ 4 \\ -10 \\ -2 \end{bmatrix} \end{array}$$

Homogeneous coordinates

$$v = \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$$

w is an arbitrary constant
 $wX = w \cdot x$

- To recover “normal” coordinates, divide the first N components by (N+1)th, w.

Homogeneous coordinates

- 3D rotation

Inhomogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = [R] \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

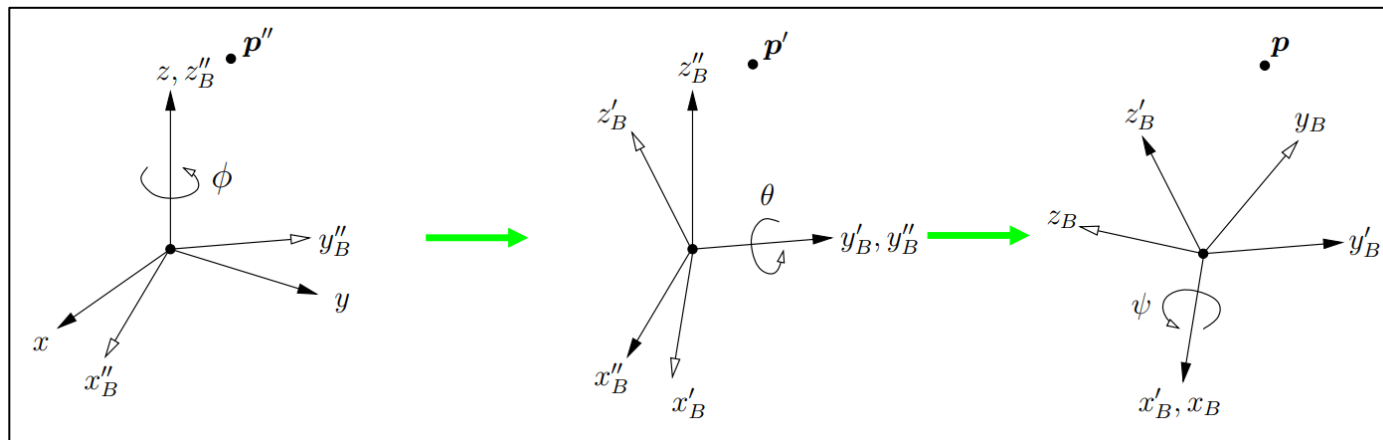
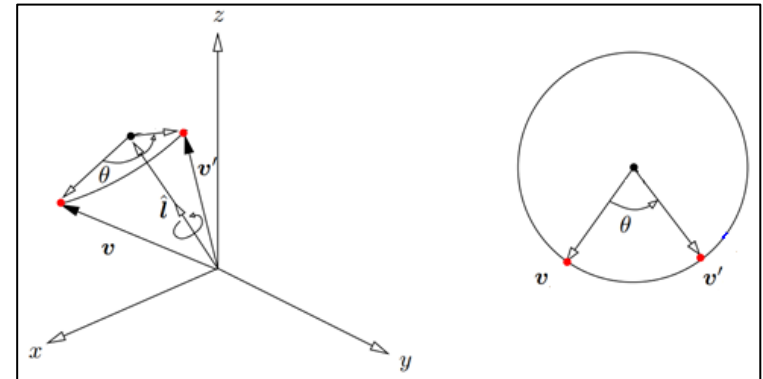
Homogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Some properties of matrix R
 - orthogonal matrix $\Rightarrow RR^T = I \Rightarrow R^{-1} = R^T$
 - the dot product of any pair of rows or any pair of columns is zero
 - normalized matrix: the squares of the elements in any row or columns sum to 1
 - the rows of R represent the coordinates in the original space of unit vectors along the coordinate axis of the rotated space
 - the columns of R represent the coordinates in the rotated space of unit vectors along the coordinate axis of the original space

(Rodrigues transform)

- The representation of rotations by 3-by-3 matrices is usually the most convenient for calculation purposes.
- The downside is that it can be difficult to intuit just what 3-by-3 matrix goes with what rotation.
- An alternate and somewhat easier-to-visualize representation for a rotation is in the form of a vector about which the rotation operates together with a single angle.



(Rodrigues transform)

- In geometry, Euler's rotation theorem states that, in three-dimensional space, any displacement of a rigid body such that a point on the rigid body remains fixed, is equivalent to a single rotation about some axis that runs through the fixed point.
- This rotation can be represented:
 - Separately: as explicit axis and angle
 - Axis = unit-length vector in 3D space
 - Angle = separate scalar value
 - Combined: as one scaled rotation vector
 - Make a unit vector axis, scaled by angle amount:
 - Vector Direction = axis of rotation
 - Vector Magnitude = angle of rotation
- The relationship between these two representations, the matrix and the vector, is captured by the Rodrigues transform.
- In linear algebra terms, the Euler's theorem states that, in 3D space, any two Cartesian coordinate systems with a common origin are related by a rotation about some fixed axis.
This also means that the product of two rotation matrices is again a rotation matrix.

(Rodrigues transform)

- The conversion from the axis-magnitude representation to a rotation matrix \mathbf{R} , and vice-versa, can be done using the Rodrigues transform.
- Let:
 - \mathbf{R} be a rotation matrix
 - \mathbf{r} be the three-dimensional vector $\mathbf{r} = [\mathbf{r}_x \ \mathbf{r}_y \ \mathbf{r}_z]$ whose direction represents the axis of rotation and whose magnitude represents the angle of rotation, θ
- OpenCV provides a function for converting from either representation to the other:
 - `void cvRodrigues2(const CvMat* src, CvMat* dst, CvMat* jacobian = NULL);`
 - to obtain vector \mathbf{r} from matrix \mathbf{R} , set **src** to be the 3-by-3 rotation matrix \mathbf{R} and **dst** to be a 3-by-1;
 - to obtain matrix \mathbf{R} from vector \mathbf{r} , set **src** to be the 3-by-1 vector \mathbf{r} and **dst** to be a 3-by-3 matrix.
 - In either case, cvRodrigues2() will do the right thing.
 - Note: the “jacobian” parameter is optional; see the manual for details.
- Note: the input/output of rotation data in several OpenCV functions is done using the axis-magnitude representation.

Homogeneous coordinates

- 3D translation

Inhomogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = [T] + \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Homogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} I & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Translation, expressed as a sum in normal coordinates is transformed into a product, in homogeneous coordinates

Homogeneous coordinates

- 3D (rotation + translation)

Homogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Using homogeneous coordinates,
Rotation and Translation can be expressed by a single matrix

Homogeneous coordinates

- Perspective projection (along Z)

$$x' = \frac{f}{z} x$$

$$y' = \frac{f}{z} y$$

Homogeneous coordinates

$$\begin{bmatrix} wx' \\ wy' \\ 0 \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- 3D scaling

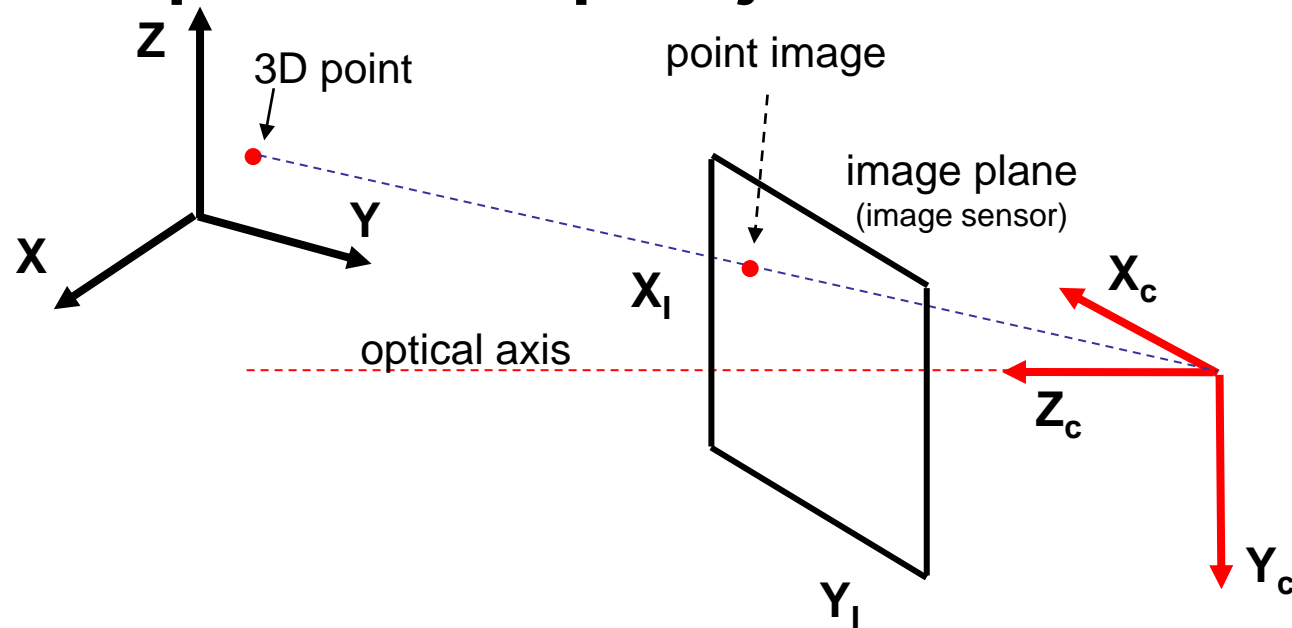
Homogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Homogeneous coordinates



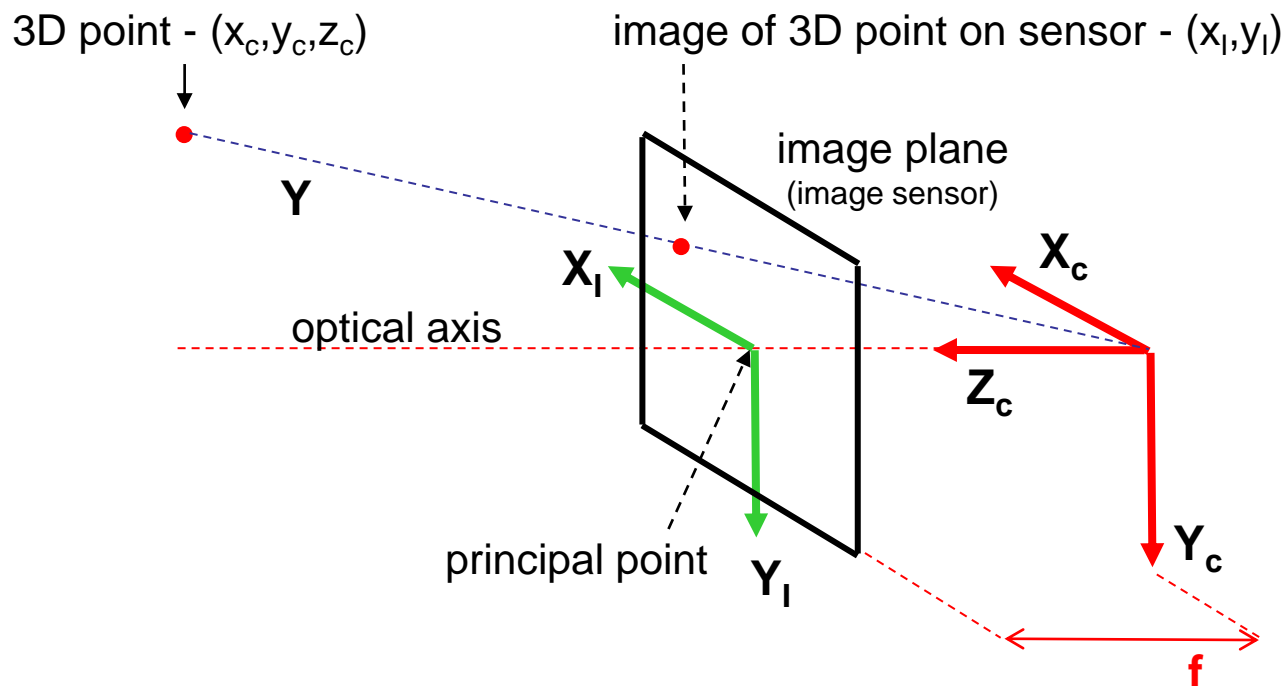
Perspective projection matrix



1. Transform world coordinates into camera coordinates

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

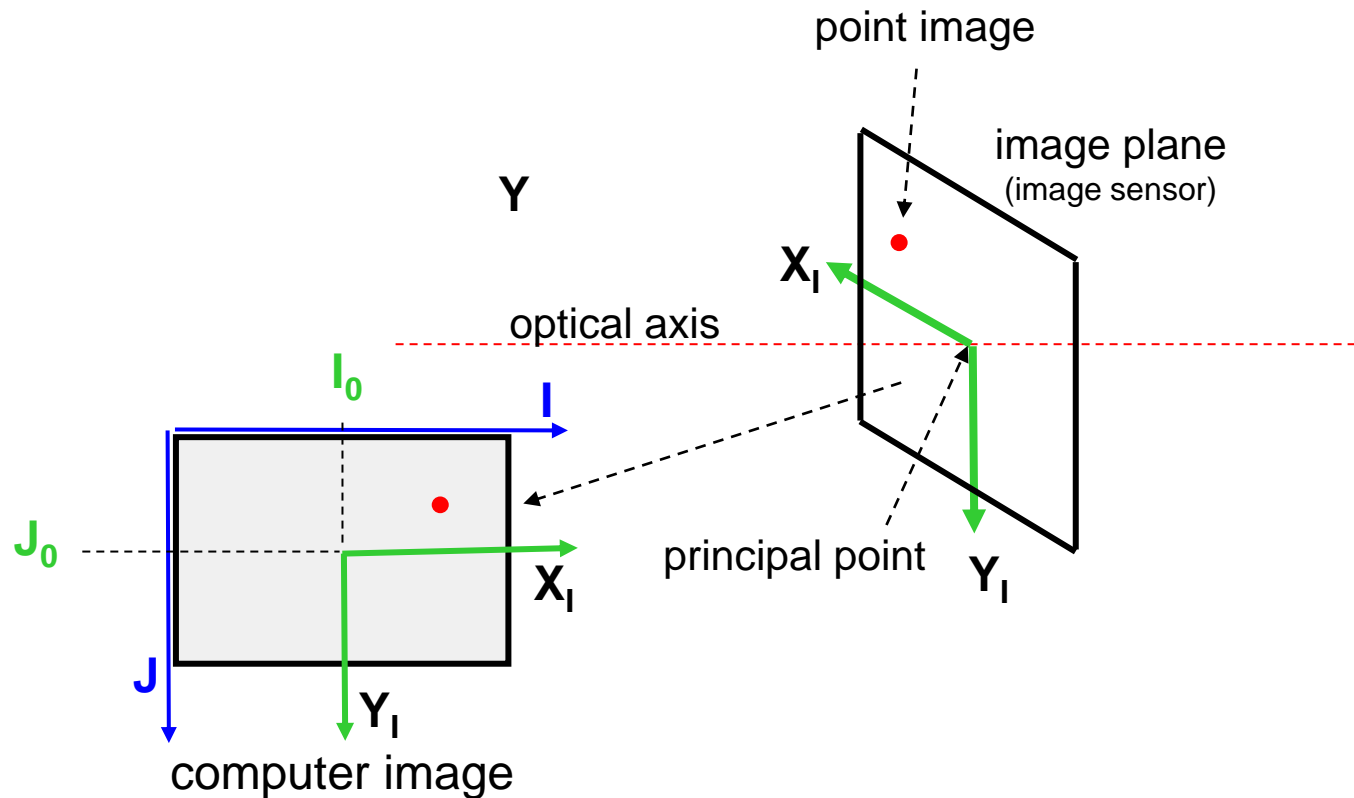
Perspective projection matrix



2. Transform camera coordinates
into sensor coordinates

$$\begin{bmatrix} wx_l \\ wy_l \\ 0 \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

Perspective projection matrix

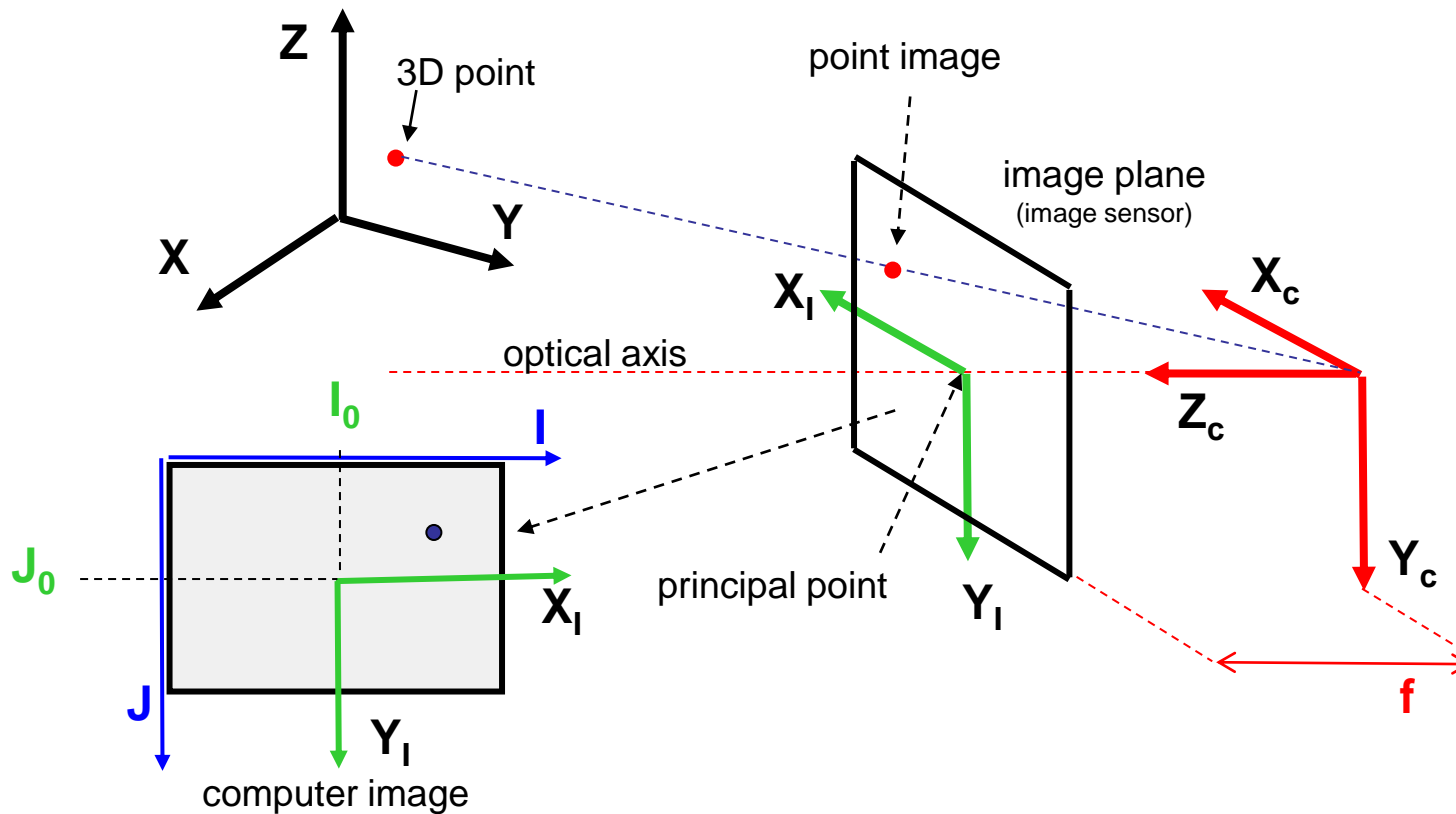


- ### 3. Transform sensor coordinates into pixel coordinates

$$\begin{bmatrix} i \\ j \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} S_i & 0 & 0 & I_0 \\ 0 & S_j & 0 & J_0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 0 \\ 1 \end{bmatrix}$$

1/Si and 1/Sj represent the pixel size (in metric units), in the X and Y directions

Perspective projection matrix



Combining all the transformations:

$$\begin{bmatrix} w_i \\ w_j \\ w \end{bmatrix} = \begin{bmatrix} S_x f & 0 & I_0 & 0 \\ 0 & S_y f & J_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} K_x & 0 & I_0 \\ 0 & K_y & J_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = [K][R | T] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$K_x = S_x f$$

$$K_y = S_y f$$

Perspective projection matrix

$$[C] = [K][R | T] = \begin{bmatrix} K_X & 0 & I_0 \\ 0 & K_Y & J_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_X \\ R_{21} & R_{22} & R_{23} & T_Y \\ R_{31} & R_{32} & R_{33} & T_Z \end{bmatrix} = \begin{bmatrix} K_X R_{11} + I_0 R_{31} & K_X R_{12} + I_0 R_{32} & K_X R_{13} + I_0 R_{33} & K_X T_X + I_0 T_Z \\ K_Y R_{21} + J_0 R_{31} & K_Y R_{22} + J_0 R_{32} & K_Y R_{23} + J_0 R_{33} & K_Y T_Y + J_0 T_Z \\ R_{31} & R_{32} & R_{33} & T_Z \end{bmatrix}$$

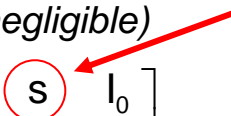
$$[C] = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} \quad \begin{bmatrix} wi \\ wj \\ w \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- $[C]$ - Perspective Projection Matrix of the camera
 - using it, the 2D image coordinates of a known 3D point can be obtained
 - also known as Direct Linear Transform (DLT) matrix
- $[K]$ - Intrinsic Parameter Matrix / Camera Matrix
 - represents the internal characteristics of the camera
- $[R / T]$ - Extrinsic Parameter Matrix
 - represents the position and orientation of the camera relatively to the world coordinate system

Camera parameters

- Intrinsic parameters

- principal point – I_0, J_0
 - intersection of the optical axis of the lens with the image sensor (pixel coord.)
 - usually not coincident with the image center
- scale factors – $K_x = S_i f$, $K_y = S_j f$
 - impossible to separate the S 's from f , unless one of them is known *a priori*
- distortion coefficients (*not considered in the previous model*)
- *skew induced by angle between sensor axes*
(*not considered in the model; usually negligible*)

$$[K] = \begin{bmatrix} K_x & s & I_0 \\ 0 & K_y & J_0 \\ 0 & 0 & 1 \end{bmatrix}$$


- Once calculated may be saved for future use, unless a zoom lens is used.

- Extrinsic parameters (camera pose)

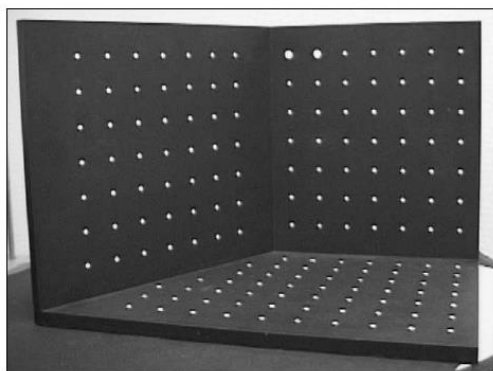
- rotation matrix
- translation vector

Camera calibration

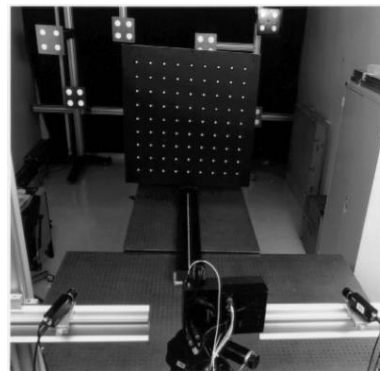
- Calibration is the process of estimating the intrinsic and extrinsic parameters of the camera.
- It can be thought of as a two stage process:
 - estimating matrix C , and
 - estimating the intrinsic and extrinsic parameters from C
- In many cases, particularly for stereo, the second stage is not necessary.
- Calibration consists of the following steps:
 - acquire an image of a set of known 3D points
 - determine the 2D image coordinates of each point
 - from each set of a 3D point and the corresponding 2D pixel a set of 2 equations results
 - establish enough correspondences to build a set of equations from which the elements of matrix C can be calculated
- How many point correspondences are needed ?

Camera calibration

- Matrix C is defined up to an arbitrary scale factor and has therefore only 11 independent entries.
- So, one of the 12 C_{ij} 's can be arbitrarily chosen
- Since each pair of 3D-2D corresponding points provides 2 equations in the unknown C_{ij} 's, at least 5.5 points (!)
 \Rightarrow 6 points are necessary
- In practice, much more than 6 points should be used
- To avoid degeneracies, calibration points must be non-coplanar



calibration target



stereo camera pair, calibration targets
and translation stage

Camera calibration procedure

DLT method

- Acquire an image of an object with at least 6 non-coplanar points whose 3D coordinates are known with high accuracy
 - in practice much more than 6 points should be used
 - points should cover all the field of view (FOV) of the camera
- Determine the 2D image coordinates of each point
- For each pair of corresponding 3D-2D points a set of 2 equations results from the perspective projection matrix

$$\begin{bmatrix} wi \\ wj \\ w \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$x_k C_{11} + y_k C_{12} + z_k C_{13} + C_{14} - i_k x_k C_{31} - i_k y_k C_{32} - i_k z_k C_{33} - i_k C_{34} = 0$$

$$x_k C_{21} + y_k C_{22} + z_k C_{23} + C_{24} - j_k x_k C_{31} - j_k y_k C_{32} - j_k z_k C_{33} - j_k C_{34} = 0$$

set of 2 equations
for a generic pair, k:
(x_k, y_k, z_k) \leftrightarrow (i_k, j_k)

Camera calibration procedure

DLT method

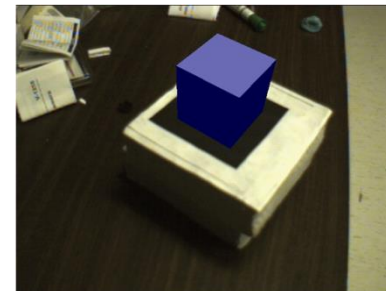
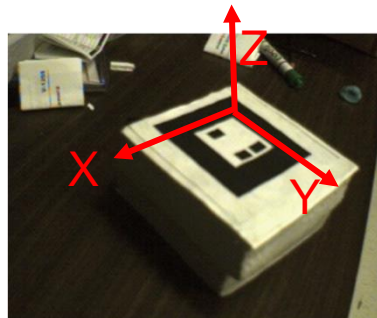
- Build a system of equations for the whole set of corresponding points:

$$\begin{bmatrix}
 x_1 & y_1 & z_1 & 1 & 0 & 0 & 0 & 0 & -i_1 x_1 & -i_1 y_1 & -i_1 z_1 & -i_1 \\
 0 & 0 & 0 & 0 & x_1 & y_1 & z_1 & 1 & -j_1 x_1 & -j_1 y_1 & -j_1 z_1 & -j_1 \\
 x_2 & y_2 & z_2 & 1 & 0 & 0 & 0 & 0 & -i_2 x_2 & -i_2 y_2 & -i_2 z_2 & -i_2 \\
 0 & 0 & 0 & 0 & x_1 & y_1 & z_1 & 1 & -j_2 x_2 & -j_2 y_2 & -j_2 z_2 & -j_2 \\
 \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
 \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
 \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
 x_N & y_N & z_N & 1 & 0 & 0 & 0 & 0 & -i_N x_N & -i_N y_N & -i_N z_N & -i_N \\
 0 & 0 & 0 & 0 & x_N & y_N & z_N & 1 & -j_N x_N & -j_N y_N & -j_N z_N & -j_N
 \end{bmatrix}
 \begin{bmatrix}
 C_{11} \\
 C_{12} \\
 C_{13} \\
 C_{14} \\
 C_{21} \\
 C_{22} \\
 C_{23} \\
 C_{24} \\
 C_{31} \\
 C_{32} \\
 C_{33} \\
 C_{34}
 \end{bmatrix}
 =
 \begin{bmatrix}
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0
 \end{bmatrix}$$

- Imposing some restriction, solve this homogeneous system of equations:
 - $C_{34}=1$,
and solve using the least-squares method or
 - $\| [C_{31} \ C_{32} \ C_{33}] \| = \| [R_{31} \ R_{32} \ R_{33}] \| = 1$,
and solve using Lagrange multipliers method
- Once the perspective projection matrix, C , is obtained,
it is possible to obtain the intrinsic and extrinsic parameters from C .

Camera pose estimation

- Many times, the intrinsic parameters are kept constant while the extrinsic parameters are varying (ex: moving camera, but not zooming)
- In those cases, to update the Perspective Projection Matrix, only the extrinsic parameters need to be updated
- Camera Pose (rotation + translation) estimation algorithms
 - From 3D-2D points correspondences:
 - DLT algorithm + Extraction of K , R , T
 - not useful in the above situation
 - P3P algorithm (PnP – Perspective n Point problem) – OpenCV: `solvePNP()` and `solvePnPRansac()`
 - POSIT (\Rightarrow 4 non-coplanar 3D points) – OpenCV: `cvPOSIT()`
 - Non-linear minimization
 - From a planar structure
 - homography between, at least, 4 points on a plane and the corresponding image points
 - Coplanar POSIT
 - PnP and EPnP



Camera pose estimation using an homography

- We have seen that the Perspective Projection Matrix $[C]$ that establishes the relationship between the 3D coordinates of a point in space and its 2D coordinates

$$\begin{bmatrix} wi \\ wj \\ w \end{bmatrix} = [C] \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

can be written as:

$$[C] = \begin{bmatrix} K_x & 0 & I_0 & 0 \\ 0 & K_y & J_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} K_x & 0 & I_0 \\ 0 & K_y & J_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \end{bmatrix} = [K][R \quad T]$$

Camera pose estimation using an homography

- When we acquire an image of a plane to which we associate the 3D world coordinate system, so that the points on the plane verify the equation $z=0$, we have

$$\begin{bmatrix} wi \\ wj \\ w \end{bmatrix} = [K] \cdot [R \quad T] \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} K_x & 0 & I_0 \\ 0 & K_y & J_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_{11} & R_{12} & T_x \\ R_{21} & R_{22} & T_y \\ R_{31} & R_{31} & T_z \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = [K] \cdot [r_1 \quad r_2 \quad t] \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = [H] \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where $[H] = [K] \cdot [r_1 \quad r_2 \quad t]$

represents an homography between the points on the plane and the points on the image.

- If we calculate the homography $[H]$ and we know matrix $[K]$ that contains the intrinsic camera parameters we can calculate the pose of the camera: $[P] = [r_1 \quad r_2 \quad t]$

$$[H] = [K] \cdot [P]$$

$$[K]^{-1} \cdot [H] = [K]^{-1} \cdot [K] \cdot [P]$$

$$[P] = [K]^{-1} \cdot [H] = [p_1 \quad p_2 \quad p_3] = [r_1 \quad r_2 \quad t]$$

Camera pose estimation using an homography

- At first, one could be tempted to say that

$$r_1 = p_1$$

$$r_2 = p_2$$

$$t = p_3$$

- However, taking into account that matrix $[H]$ is defined up to an arbitrary scale factor, it is probable that p_1 and p_2 are not in normalized form:

$$\|p_1\| = \|r_1\| = 1 \quad \|p_2\| = \|r_2\| = 1 \quad \|p_1 \cdot p_2\| = \|r_1 \cdot r_2\| = 0$$

- In order to normalize $[P]$ we must divide its members by a normalization factor λ :

$$[\lambda] = \sqrt{\|p_1\| \|p_2\|}$$

NOTE: the sign of λ can be positive or negative and that will influence the sign of the elements of the translation vector;
the choice of the adequate sign must take into account the position of the camera relatively to the reference plane

- Then, we calculate:

$$r_1 = \frac{p_1}{\lambda} \quad r_2 = \frac{p_2}{\lambda} \quad t = \frac{p_3}{\lambda}$$

NOTE: Do not forget that p_1, p_2, p_3, r_1, r_2 and t are, in fact, vectors.

In the following step we shall use vectorial notation to represent r_1 and r_2

Camera pose estimation using an homography

- However, this step does not yet guarantee that $\vec{r}_1 \cdot \vec{r}_2 = 0$
as it may happen that r_1 and r_2 are not exactly perpendicular.

- So, a second correction is necessary:

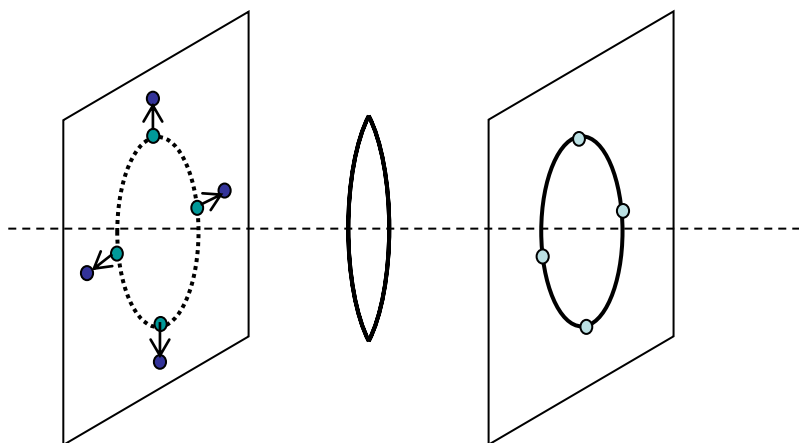
$$\begin{aligned}\vec{a} &= \vec{r}_1 + \vec{r}_2 \\ \vec{b} &= \vec{r}_1 \times \vec{r}_2 \\ \vec{d} &= \vec{a} \times \vec{b}\end{aligned}\quad \begin{aligned}\vec{r}_{1f} &= \frac{\vec{a}}{\|\vec{a}\|} + \frac{\vec{d}}{\|\vec{d}\|} \\ \vec{r}_{2f} &= \frac{\vec{a}}{\|\vec{a}\|} - \frac{\vec{d}}{\|\vec{d}\|}\end{aligned}$$

- Vectors r_{1f} and r_{2f} can still not be in normalized form,
so it is necessary to normalize them

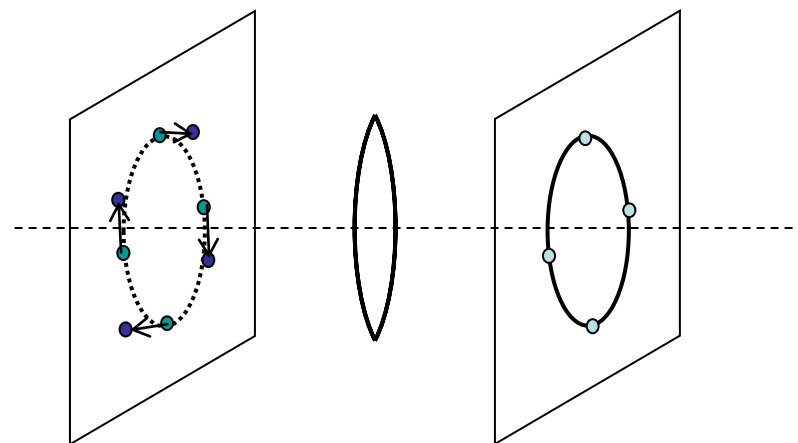
$$\begin{aligned}\vec{r}_{1f} &= \frac{\vec{r}_{1f}}{\|\vec{r}_{1f}\|} \\ \vec{r}_{2f} &= \frac{\vec{r}_{2f}}{\|\vec{r}_{2f}\|}\end{aligned}$$

- Finally, we calculate $\vec{r}_{3f} = \vec{r}_{1f} \times \vec{r}_{2f}$

Lens geometric distortions



Radial distortion

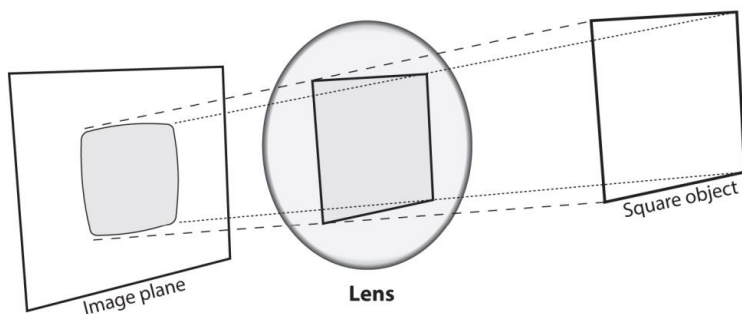


Tangential distortion

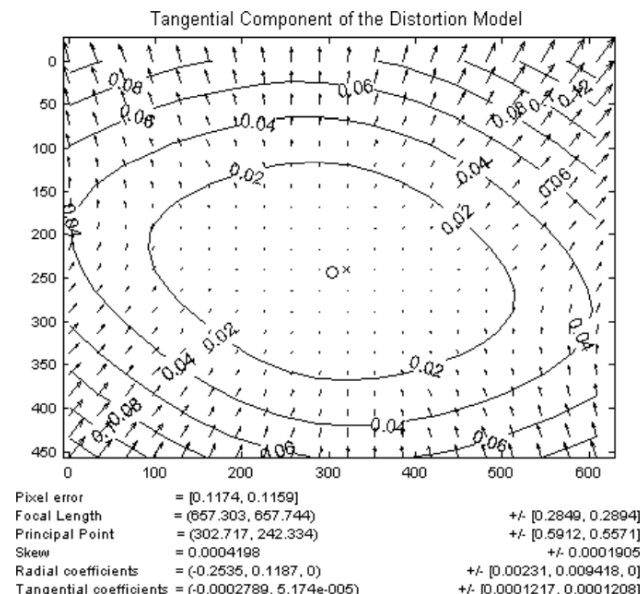
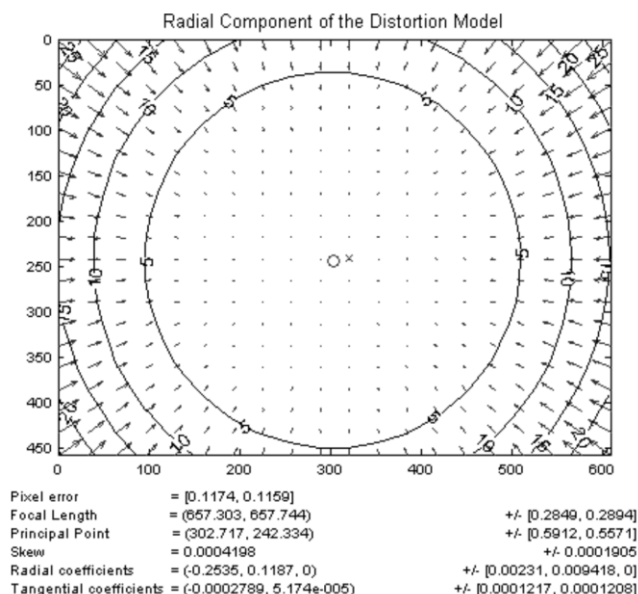
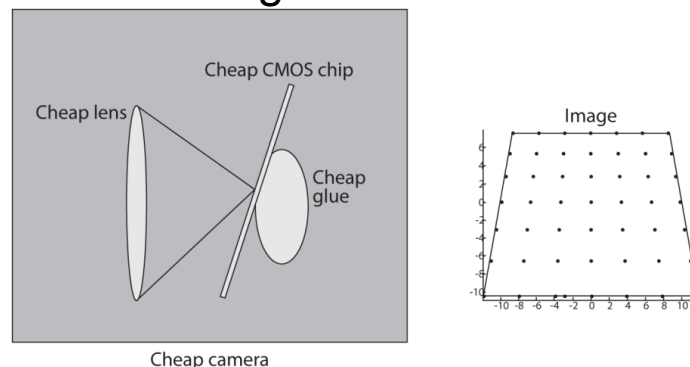
- Radial:
 - is a side effect of the round glass elements within a lens and the effect of bending light more/less near the edges of the lens than we encounter near the center of the lens
- Tangential:
 - is due to imperfect alignment or imperfect centering of the lens components and other manufacturing defects in a compound lens
- Rectify with geometric camera calibration

Lens geometric distortions

Radial distortion



Tangential distortion



Plots for a particular camera lens: the arrows show where points on an external rectangular grid are displaced in a radially (on the left) or tangentially (on the right) distorted image

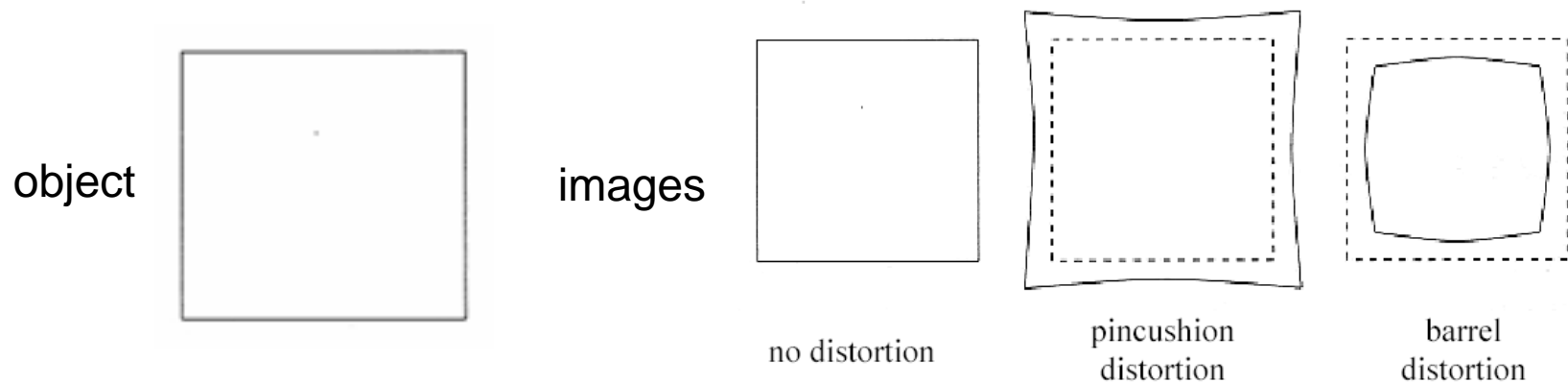
(source: [Learning OpenCV book](#))

Lens geometric distortions

- Radial distortion



Radial lens distortions: (a) barrel; (b) pincushion; (c) pincushion (fisheye lens)



Radial distortion:

- causes straight lines to appear curve;
- becomes larger the farther points are from the center of the image.

Lens geometric distortions

- Radial distortion can be represented as follows:

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

- Tangential distortion can be represented as follows:

$$x_{distorted} = x + [2p_1 xy + p_2 (r^2 + 2x^2)]$$

$$y_{distorted} = y + [p_1 (r^2 + 2y^2) + 2p_2 xy]$$

- In short, we need to find five parameters, known as distortion coefficients given by:

$$\text{Distortion coefficients} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

- *Note: this is the order in which OpenCV places them in a 5x1 matrix*

Camera calibration methods

- Several methods have been proposed:
 - DLT (Direct Linear Transform) method (*the previously referred method*)
 - *Karara & Abdel-Aziz, Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry, 1971*
 - Heikkilä & Silvén method
 - *A Four-step Camera Calibration Procedure with Implicit Image Correction, CVPR97*
 - Zhang method
 - *Flexible Camera Calibration by Viewing a Plane from Unknown Orientations, ICCV99*
 - ...
- Some of these methods allow for the determination of intrinsic and extrinsic parameters, as well as distortion coefficients.
- See "[Camera Calibration Toolbox for Matlab](#)" for extensive list of calibration methods and implementations
 - http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/parameters.html
- OpenCV documentation
 - http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html
 - https://docs.opencv.org/3.4/d4/d94/tutorial_camera_calibration.html
 - https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html
 - https://docs.opencv.org/3.4/d9/d0c/group_calib3d.html#ga93efa9b0aa890de240ca32b11253dd4a
 - https://github.com/opencv/opencv/blob/3.4/samples/cpp/tutorial_code/calib3d/camera_calibration/camera_calibration.cpp
 - https://www.mrpt.org/downloads/camera-calibration-checker-board_9x7.pdf

Evaluating the model

Reprojection error

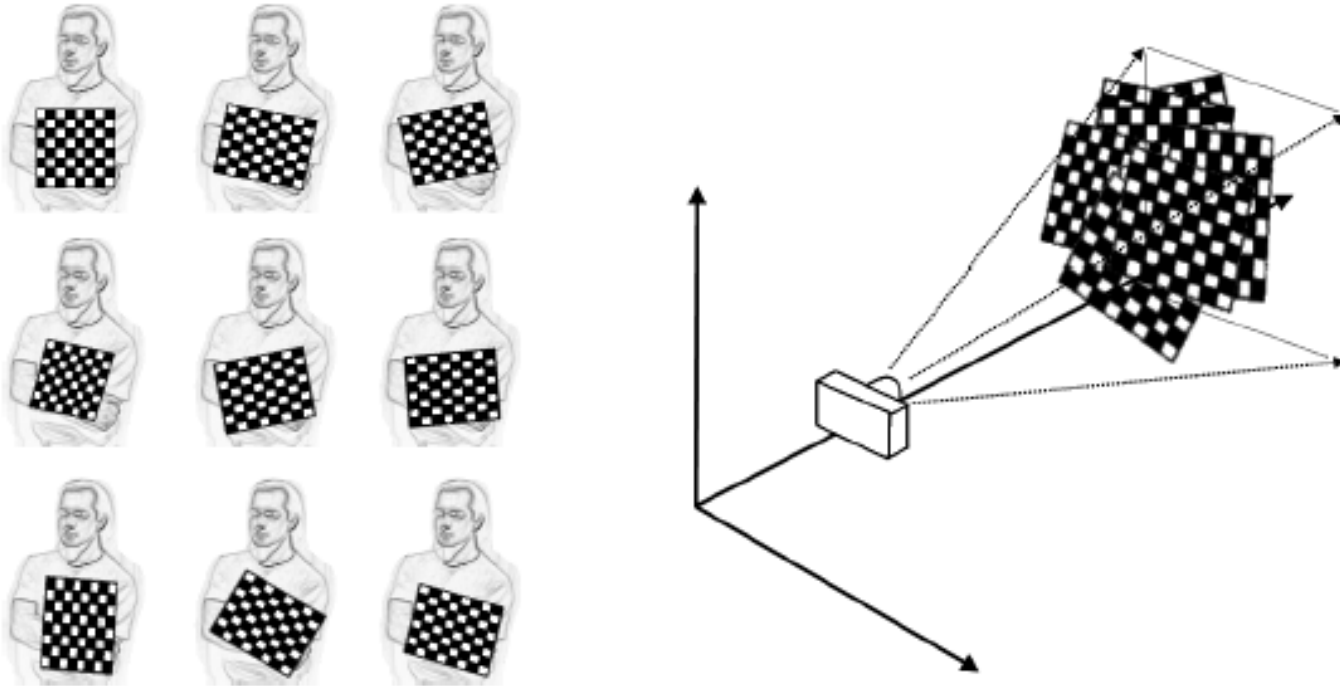
- The reprojection error is an indicator of calibration accuracy.
- The RMS reprojection error is the mean value of the squares of the distances between the computed (projected) locations of the 3D points onto the image plane and the actual location of the corresponding points on the original image.
- To find the error we calculate the arithmetical mean of the errors calculated for all the calibration images.
- Given the intrinsic, extrinsic (rotation and translation) and distortion parameters, we first transform the object point to image point (OpenCV, using **projectPoints()**) then we calculate the absolute distance between what we got with the transformation and the corresponding corner detected by the corner finding algorithm.

OpenCV - camera calibration

- **findChessboardCorners**
- **calibrateCamera()**
 - Finds the camera intrinsic (**cameraMatrix**) and extrinsic parameters (**rvecs** & **tvecs**) from several views of a calibration pattern.
 - overloaded function; has a simple version with less parameters
 - returns the overall RMS re-projection error
- **decomposeProjectionMatrix()**
- **solvePnP() & solvePnPRansac()**
- **undistortPoints() & undistort()**
- **projectPoints()**
- **findHomography()**
- **decomposeHomographyMat()**
- ... and many other functions

https://docs.opencv.org/master/d9/d0c/group__calib3d.html

OpenCV - camera calibration



Images of a chessboard being held at various orientations (left) provide enough information to completely solve for the locations of those images in global coordinates (relative to the camera) and the camera intrinsics

source: [Learning OpenCV 3](#), Adrian Kaehler & Gary Bradski

"Inverting" the perspective transformation

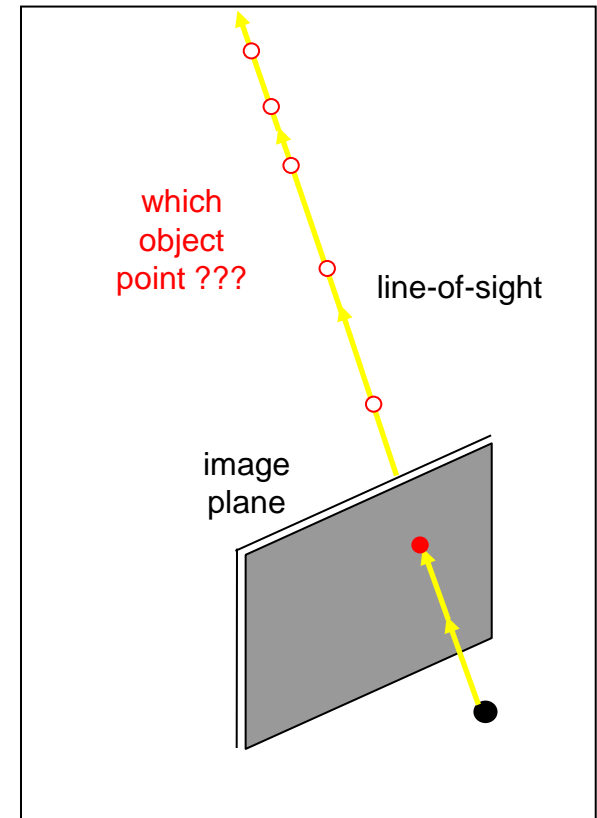
- Matrix C can't be inverted
 - it is, obviously, not possible to obtain the coordinates, (x,y,z) , of a 3D point, given the coordinates, (i,j) , of the corresponding 2D pixel

- Given matrix C
$$\begin{bmatrix} wi \\ wj \\ w \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

and the coordinates of a pixel, (i,j) , the following set of equations can be obtained:

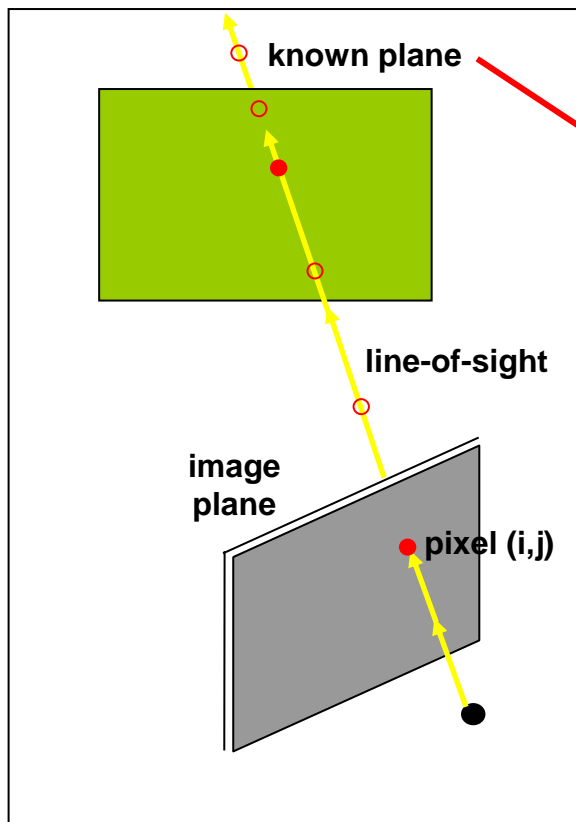
$$\begin{aligned} (C_{31}i - C_{11}) \cdot x + (C_{32}i - C_{12}) \cdot y + (C_{33}i - C_{13}) \cdot z &= C_{14} - C_{34}i \\ (C_{31}j - C_{21}) \cdot x + (C_{32}j - C_{22}) \cdot y + (C_{33}j - C_{23}) \cdot z &= C_{24} - C_{34}j \end{aligned}$$

- These are the equations of 2 planes whose intersection determines the line-of-sight of pixel (i,j) .



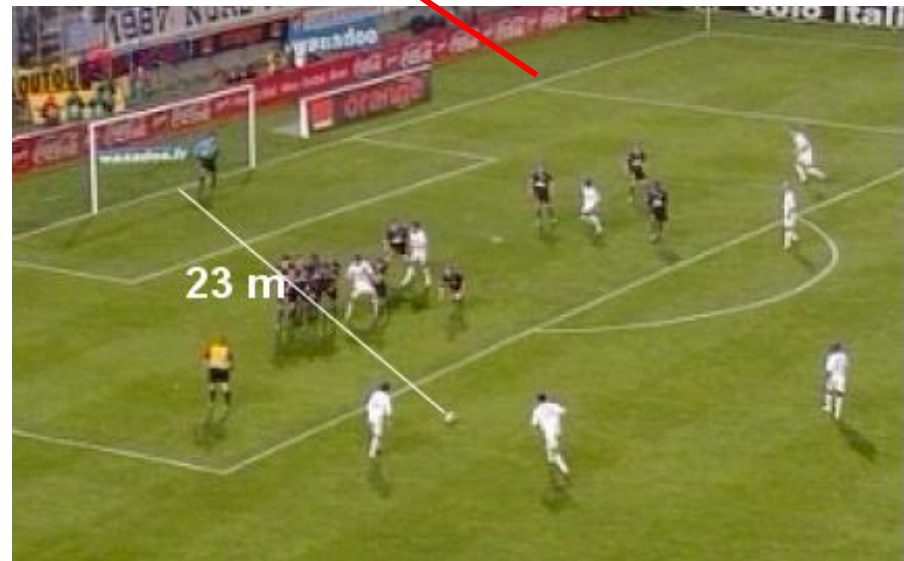
"Inverting" the perspective transformation

- If one knows that the scene point lies on a given 3D plane, its 3D coordinates can be obtained with a single image



$$\begin{cases} (C_{31}i - C_{11}) \cdot x + (C_{32}i - C_{12}) \cdot y + (C_{33}i - C_{13}) \cdot z = C_{14} - C_{34}i \\ (C_{31}j - C_{21}) \cdot x + (C_{32}j - C_{22}) \cdot y + (C_{33}j - C_{23}) \cdot z = C_{24} - C_{34}j \\ A \cdot x + B \cdot y + C \cdot z = D \end{cases}$$

ex: $z=0$ ($\rightarrow A=0, B=0, C=1, D=0$)



(Homography)

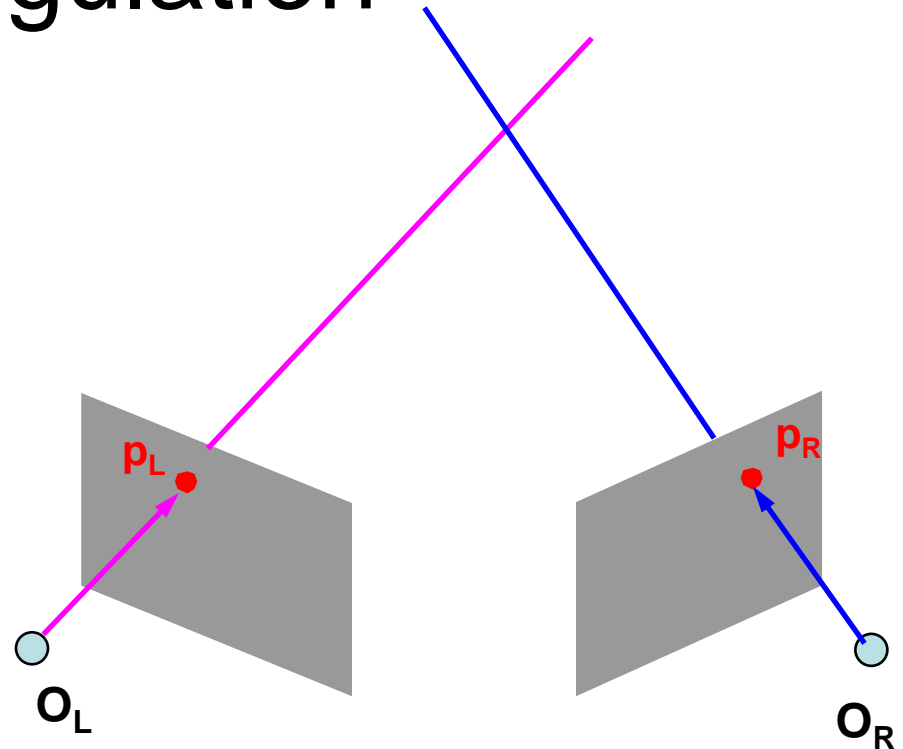
$$\begin{bmatrix} w_i \\ w_j \\ w \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} \quad \leftarrow \boxed{z = 0}$$

$$\begin{bmatrix} w_i \\ w_j \\ w \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homography
between 2 planes

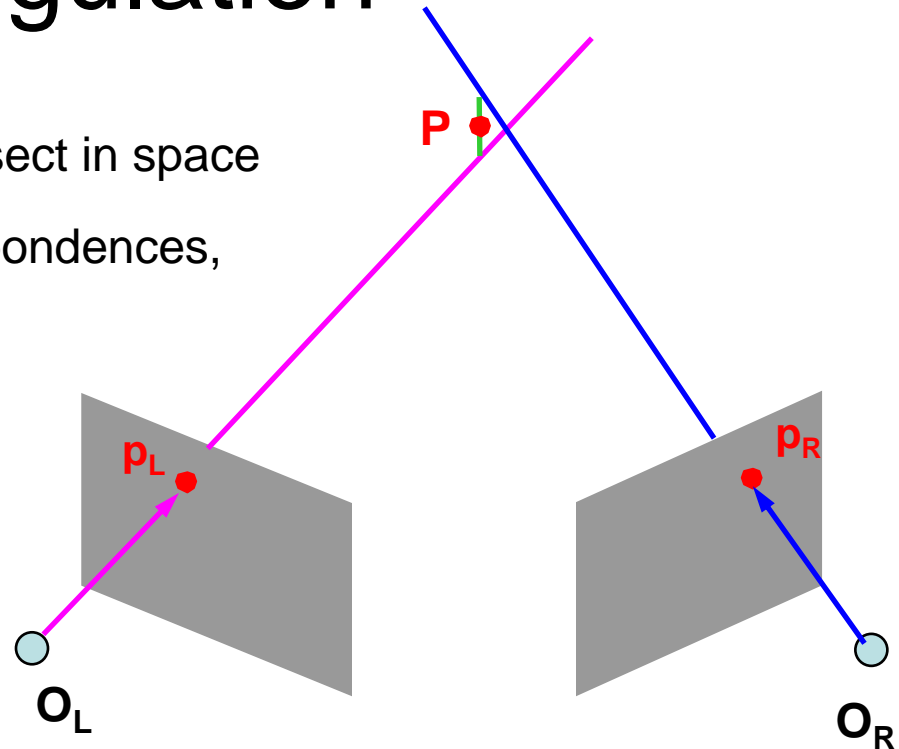
3D reconstruction by triangulation

- Assuming that
 - cameras are calibrated
 - correspondences between points of the stereo pair are established
- For each pair of corresponding pixels
 - determine the line-of-sight associated with each pixel
 - intersect the 2 lines-of-sight



3D reconstruction by triangulation

- Problem:
 - the 2 rays will not actually intersect in space due to errors in calibration and correspondences, and pixelization
- Solution:
 - find a point in space with minimum distance (in the least squares sense) from both rays



$$\begin{aligned}
 \text{Left line-of-sight} &\rightarrow \begin{cases} (C_{L_{31}}i - C_{L_{11}}) \cdot x + (C_{L_{32}}j - C_{L_{12}}) \cdot y + (C_{L_{33}}i - C_{L_{13}}) \cdot z + = C_{L_{14}} - C_{L_{34}}i \\ (C_{L_{31}}j - C_{L_{21}}) \cdot x + (C_{L_{32}}j - C_{L_{22}}) \cdot y + (C_{L_{33}}j - C_{L_{23}}) \cdot z + = C_{L_{24}} - C_{L_{34}}j \end{cases} \\
 \text{Right line-of-sight} &\rightarrow \begin{cases} (C_{R_{31}}i - C_{R_{11}}) \cdot x + (C_{R_{32}}i - C_{R_{12}}) \cdot y + (C_{R_{33}}i - C_{R_{13}}) \cdot z + = C_{R_{14}} - C_{R_{34}}i \\ (C_{R_{31}}j - C_{R_{21}}) \cdot x + (C_{R_{32}}j - C_{R_{22}}) \cdot y + (C_{R_{33}}j - C_{R_{23}}) \cdot z + = C_{R_{24}} - C_{R_{34}}j \end{cases}
 \end{aligned}$$

Practical work

- Calibrate camera
 - analyze the returned parameters and the reprojection error
- Undistort an acquired image.
- Estimate the pose of a camera, relatively to a fixed planar target.
- Measure the distance between 2 points marked on the target using an image of the target.
- Project on the image a virtual line in 3D world;
the line must be perpendicular to the planar target
and connect the center of the target with a point 20 cm above the target.